

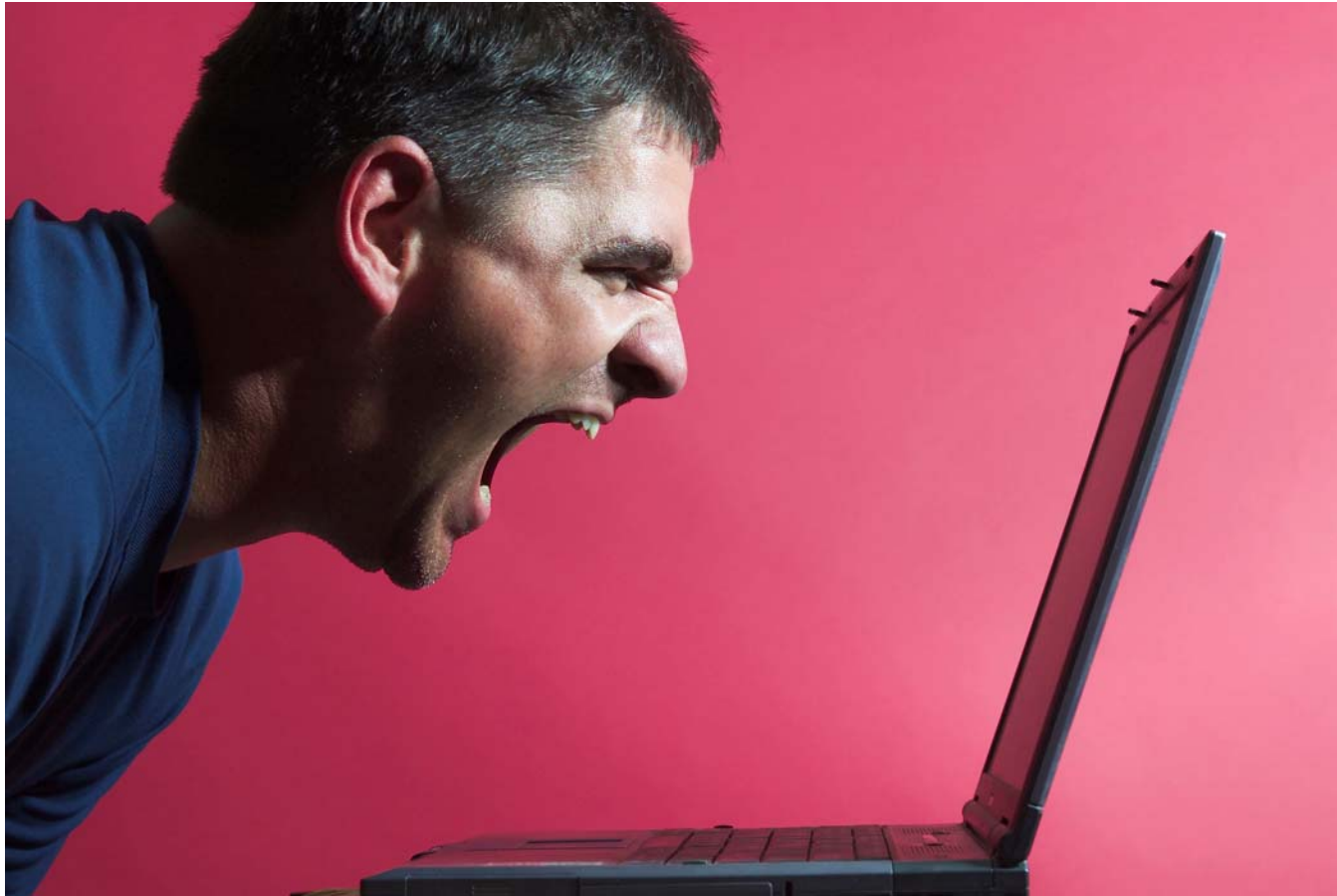
Phys 7411: Languages and Compilers

Karen Tomko and Juana Moreno

ktomko@osc.edu or moreno@phys.lsu.edu

Spring 2009

Is this your reaction to MPI?



What are the options

- Automatic parallelization
- New Languages
- Augment commonly used languages

Automatic Parallelization

- Lots of research work in 1980s, 1990s
 - Identify parallel loop nests in array based code
 - Based on dependency analysis (more on this)
 - Determine what references (variable reads or writes) access the same memory location
 - Determine ordering requirements between statements or loop iterations

Dependence analysis example

```
Real A(1000,1000)
```

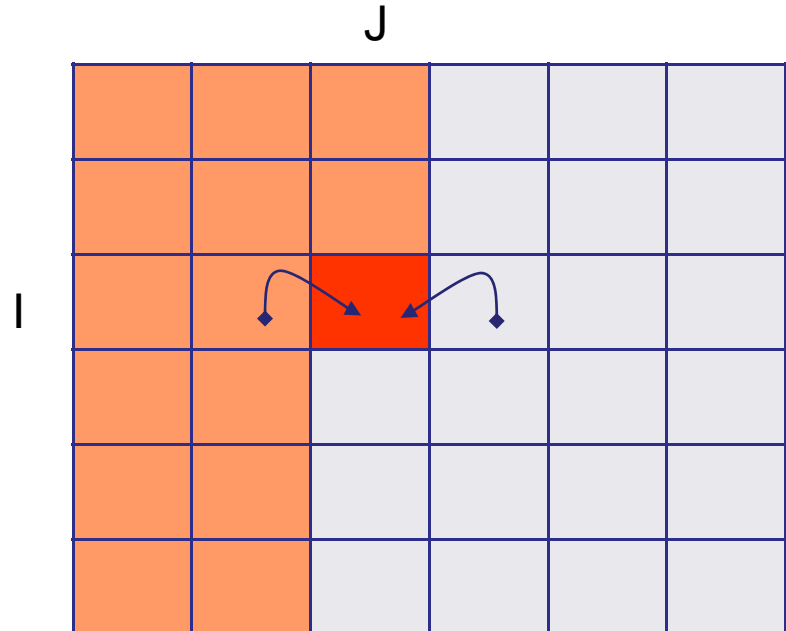
```
DO J = 2, N-1
```

```
  DO I = 1, N
```

```
    A(I,J) = (A(I, J+1) + 2*A(I,J) + A(I, J-1))*0.25
```

```
  ENDDO
```

```
ENDDO
```



Automatic Parallelization

- Difficulties
 - Compiler's scope of analysis limited to a few loop nests
 - Compilation must be fast
 - Pointers defy analysis
 - Its easy to generate a slow parallel program

New Languages

- Some Parallel Languages that never made it
 - Sisal
 - Cilk
 - Occam
 - High Performance Fortran (HPF)
- Some that could (more about these later)
 - Unified Parallel C (UPC)
 - Co-array Fortran

Augmented languages

- Completely new languages rarely catch on
- So add parallel capabilities to an existing language
 - Add special comments
 - Compiler directives or hints
 - Add new control constructs or data representations

High Performance Fortran

- Data parallel language for distributed memory systems (e.g. Linux clusters)
- What's data parallelism?
 - Mechanism to specify how data is partitioned across processors
`REAL A(1000,1000)`
`!HPF$ DISTRIBUTE A(Block, *)`
 - Owner Computes Rule
 - Processor owning a data element, execute the code which updates that elements value
 - Parallelization hints
`!HPF$ INDEPENDENT`

HPF example

```
Real A(1000,1000)
```

```
!HPF$ DISTRIBUTE A(Block, *)
```

```
DO J = 2, N-1
```

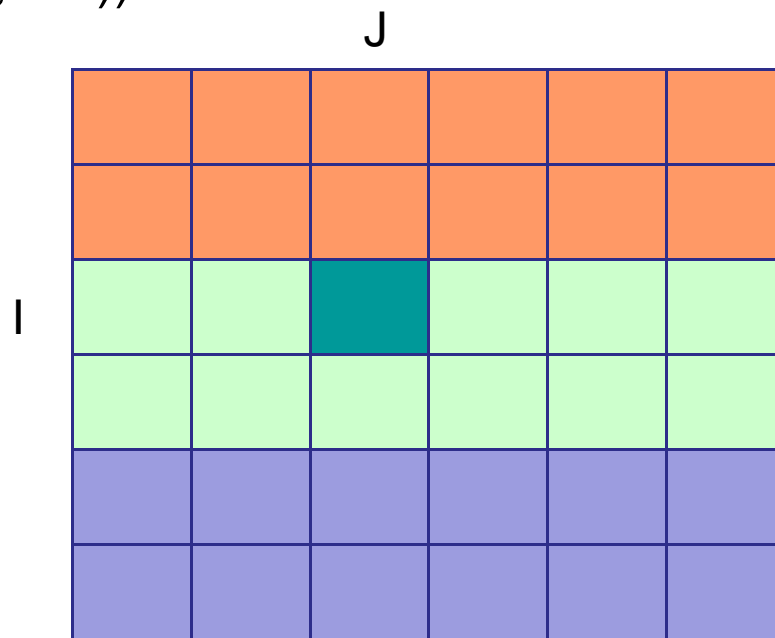
```
!HPF$ INDEPENDENT
```

```
DO I = 1, N
```

```
  A(I,J) = (A(I, J+1) + 2*A(I,J) + A(I, J-1))*0.25
```

```
ENDDO
```

```
ENDDO
```

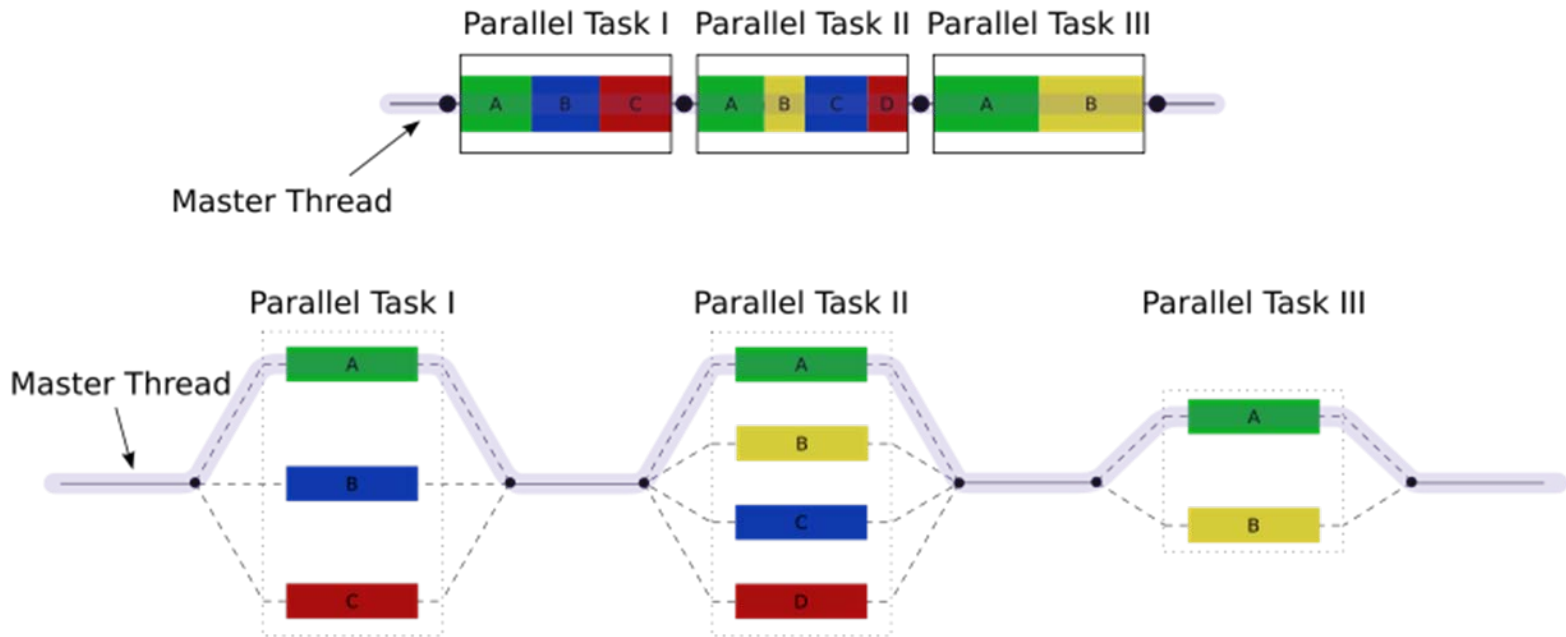


OpenMP

- Parallel programming for Shared Memory programming
- Language bindings for C, C++, Fortran, Fortran95
- Many compilers now recognize OpenMP
 - PGI compilers
 - Intel compilers

OpenMP

- Basic model



From <http://en.wikipedia.org/wiki/OpenMP>

OpenMP

- Directives to
 - Specify parallelism
`C$OMP PARALLEL DO`
 - Specify whether variables are shared or private
`C$OMP PARALLEL DO PRIVATE(I,J, WORK)`
 - Specify reduction variables
`C$OMP PARALLEL DO REDUCTION(MAX:
RelError)`

OpenMP example

```
Real A(1000,1000)
```

```
DO J = 2, N-1
```

```
C$OMP PARALLEL DO
```

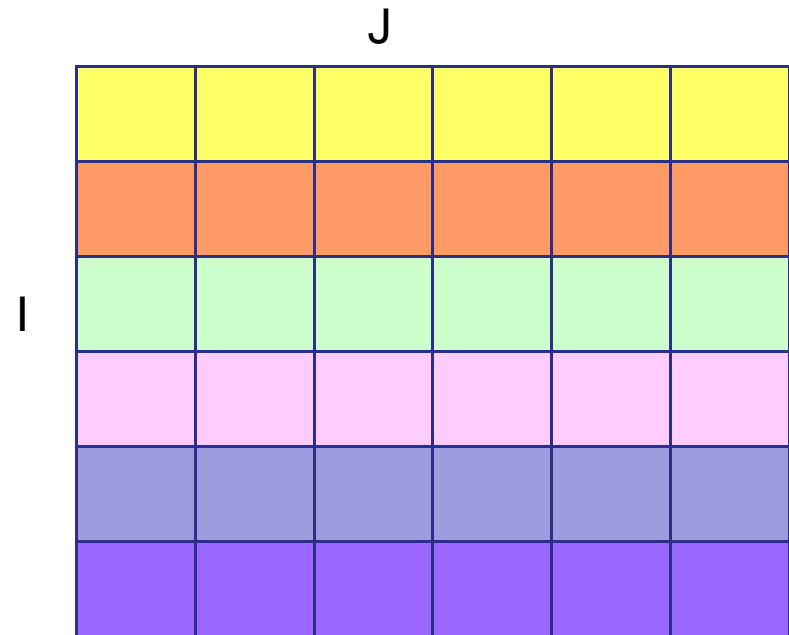
```
DO I = 1, N
```

```
  A(I,J) = (A(I, J+1) + 2*A(I,J) + A(I, J-1))*0.25
```

```
ENDDO
```

```
ENDDO
```

Loop Iteration variables are private by default



A practical option

- Parallel libraries
 - ScaLAPACK – dense linear algebra
 - Petsc – solution of PDEs
 - ARPACK – sparse eigenvalues
 - ... many more.....

Happy Mardi Gras!
See you next Friday